EQUIVALENT SYSTEM MASS ANALYSIS OF ASTRONAUT DIETS FOR LONG-DURATION SPACE MISSIONS

Presented to the Faculty of the Graduate School

of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Biological and Environmental Engineering

by

Charlotte Jenna Kiang

January 2017

ACKNOWLEDGMENTS

The author would like to acknowledge Dr. Jean B. Hunter for her mentorship and advisement throughout this project, as well as Dr. C. Lindsay Anderson for her help with MATLAB code development. Furthermore, the author would like to thank her friends and family, as well as all members of the Cornell Microgravity Research Team and Bioastronautics and Human Performance Laboratory, for the support and camaraderie they have provided throughout her master's degree.

TABLE OF CONTENTS

LIST OF ACRONYMS

BVAD	Baseline Values and Assumptions Document
ECLSS	Environmental Control and Life Support System
ESM	Equivalent System Mass
HEOMD	Human Exploration and Operations Mission Directorate
HERA	Human Exploration Research Analog
HI-SEAS	Hawai'i Space Exploration Analog and Simulation
ISS	International Space Station
LEO	Low-Earth Orbit
MATLAB	Matrix Laboratory
NASA	National Aeronautics and Space Administration
USDA	United States Department of Agriculture

I. INTRODUCTION

As of 2016, the National Aeronautics and Space Administration's (NASA's) Human Exploration and Operations Mission Directorate (HEOMD) is working towards manned missions to destinations beyond low-Earth orbit (LEO), with the ultimate goal of sending humans to Mars by 2030. [1] In contrast to previous manned missions to the Moon and International Space Station (ISS), where astronauts have had easy access to resources from Earth throughout the mission, long-duration missions to Mars will require crew members to reside in man-made habitats on the surface of Mars for extended periods of time. As a result, the design of a Mars mission must take into account how humans can efficiently sustain themselves in these largely closed-loop habitats. Since humans require adequate nutrition and caloric intake to function and maintain basic standards of health, planning for long-duration space missions requires an understanding of the effect that astronaut diets and food systems have on the overall mission.

Whereas astronauts on missions to the ISS can currently pack their own food from Earth and rely on cargo resupply missions for replenishment, spacecraft launch mass restrictions dictate that astronauts on Mars missions must grow, process and cook their own food in space in order to survive. (Given the high costs of a mission from Earth to Mars, using cargo resupply as the sole source of food is not a viable option for Mars surface missions.) [2] Thus, assembling the optimal diet for astronauts on these missions requires an analysis of the resources required to produce all desired crops and ingredients, and eventually meals. This paper details one possible approach to this analysis using the concept of Equivalent System Mass (ESM), which relates the mass, volume, power and cooling requirements, resupply needs, and crew time needs of a crop or ingredient to launch mass, allowing for determination of which crops and ingredients have the lowest cost to the mission. [3]

II. BACKGROUND

A. ESM RATIONALE

The rationale for using mass as the primary measure of cost for a space mission can be traced back to Tsiolkovsky's Rocket Equation, which states that the maximum change in velocity for a space vehicle depends on the ratio between its mass without any fuel (hereafter referred to as its "empty mass") and its mass when fully fueled (hereafter referred to as its "fueled mass"). Specifically, the equation is as follows:

$$\Delta v = v_e \ln\left(\frac{m_f}{m_e}\right)$$

Where:

Δv = maximum change in the space vehicle's velocity

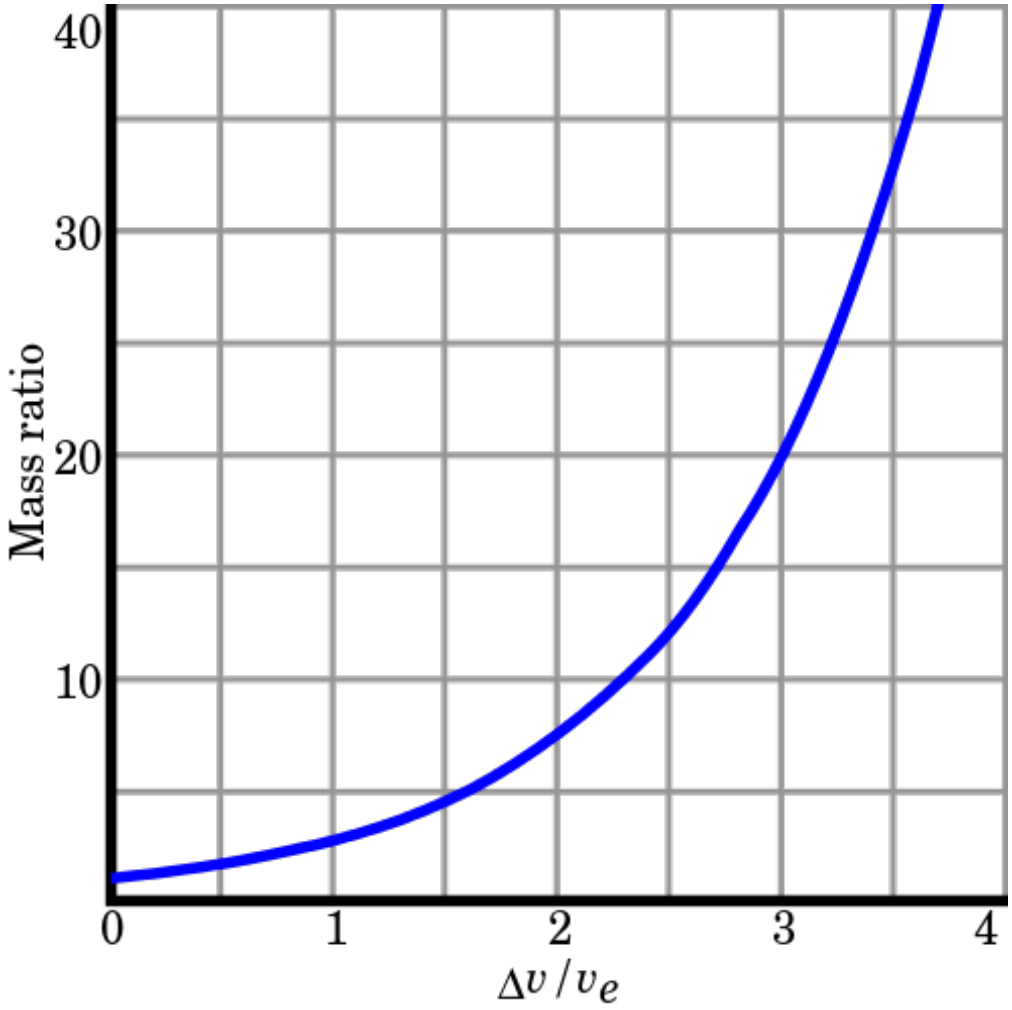
v_e = effective exhaust velocity of the vehicle

m_f = fueled mass of the vehicle

m_e = empty mass of the vehicle (equal to the vehicle's structural mass plus its payload mass) [4]

The relationship between the vehicle's fueled-to-empty mass ratio (hereafter referred to as its "mass ratio") and its ratio of change in velocity to effective exhaust velocity (hereafter referred to as its "velocity ratio") is thus exponential, as shown below in Figure I.

Figure I: Relationship between vehicle mass ratio and velocity ratio



While Tsiolkovsky's Rocket Equation is an ideal equation that does not take into account any external forces acting on the space vehicle, it demonstrates that maximizing the ratio between fueled and empty mass is critical to achieving the high maximum velocity necessary for deep space missions. Since the structural mass of any given space vehicle (such as the Orion spacecraft NASA currently plans to use for missions beyond LEO) is fixed, this means that the most efficient mission design will be one that minimizes payload mass. Thus, it makes sense to use ESM as the primary basis of comparison between food options, as it allows direct comparison between all options in terms of mass.

One criticism of ESM worth noting is that the calculations do not take into account redesign costs – for example, if an existing system has a greater ESM cost to the mission than a theoretical system that does not yet exist or is incompatible with other aspects of the overall mission, the cost of designing, manufacturing, and implementing the new system may exceed the difference in ESM cost between the two systems. In such a scenario, the existing system could be the better option. [5] Therefore, it should be noted that ESM calculations enable a first pass analysis of the lowest cost options for a mission, but that final mission planning may require further investigation into whether potential redesigns are economical. For the purposes of this project and paper, it is assumed that few if any redesigns will be required for the systems of interest, and that redesign costs will therefore be negligible.

B. BASIC ESM CALCULATION

The foundation of this paper and project is the basic equation for calculating ESM, which is as follows:

$$ESM = M + \left(V * V_{eq}\right) + \left(P * P_{eq}\right) + \left(C * C_{eq}\right) + (CT * D * \ CT_{eq})$$

Where:

ESM = equivalent system mass value of the system of interest (kg)

M = total mass of the system (kg)

V = total pressurized volume of the system (m^3)

V_{eq} = mass equivalency factor of pressurized volume for the system (kg/m^3)

P = total power requirement for the system (electrical kW*)*

P_{eq} = mass equivalency factor for the power generation infrastructure (kg/kW)

C = cooling requirement for the system (thermal kW)

C_{eq} = mass equivalency factor for the cooling infrastructure (kg/kW)

CT = total crew time requirement for the system (crew hours/year)

D = duration of the mission (years)

CT_{eq} = mass equivalency factor of the crew time support (kg/crew hours) [3]

Thus, volume, power and cooling requirements, and crew time requirements are converted into an equivalent mass to provide a common metric of comparison. Calculation for equivalency values is outside the scope of this paper and project, and in most cases is taken from other published works such as NASA's ISS Baseline Values and Assumptions Document (BVAD). [6]

It should be noted that some ESM calculations performed for this paper and project are done using variations of this equation; however, justification for deviations from the general equation is provided, and the conceptual reasoning behind the calculations remains the same.

III. PROJECT GOALS AND METHODS

The primary goal of this project was to create a scalable and well-documented program using the Matrix Laboratory (MATLAB) programming language. The basic requirements for the program were as follows:

1. To be able to read in data on crops and food processes and calculate ESM for crops and processed ingredients based on that data,
2. Optionally, to include additional functionality for conducting sensitivity analysis of crops and ingredients to various mission parameters,

3. Optionally, to include additional functionality for reading in a menu or list of ingredients and calculating the necessary surface area to grow the menu or ingredients.

Given this set of requirements, it was determined that MATLAB would be the best platform for developing the software package. One of the primary reasons for selecting MATLAB was that it was capable of efficiently reading in large amounts of data from Excel, where much of the crop and processing data was already stored. Additionally, MATLAB is a commonly used language that runs on both Windows and Mac OS operating systems, allowing for easy collaboration with NASA colleagues and future graduate students. Finally, since a future goal for this work was to be able to perform optimizations on diets and menus, it was desirable for the chosen platform to include built-in functionality for linear optimization, which was another criterion met by MATLAB.

IV. MATLAB PROGRAM

The MATLAB program for this paper and project was designed so that all functions read in a list of crops or processes, find the appropriate spreadsheet containing data for the crop or process, and then perform the requested calculation. As a result, it is very easy to add additional crops or processes to the list of desired calculations. Furthermore, the program includes robust error-checking functionality that halts all calculations and notifies users if the data spreadsheets they have provided contain unusable or nonsensical values.

A. PROGRAM STRUCTURE

The MATLAB program consists of 5 main functions for crop and ingredient ESM calculations:

1. **CalculateCropESM.m**

   This function reads in data from crop spreadsheets and calculates the ESM for any given crop according to our formula (detailed in section IV-B).

2. **CalculateProcessedESM.m**

   This function reads in data from food processing spreadsheets and calculates the ESM for all ingredients produced by each process according to our formula (detailed in section IV-C).

3. **CropESMLoop.m**

   This function reads in a list of crops and runs CalculateCropESM.m for each crop, resulting in a list of ESM values for all crops.

4. **ProcessedESMLoop.m**

   This function reads in a list of food processes and runs CalculateProcessedESM.m for each process, resulting in a list of ESM values for all ingredients produced by the given processes.

5. **Main.m**

   This program runs all calculation loops (i.e. CropESMLoop.m and ProcessedESMLoop.m) and returns a consolidated list of ESM values for all crops and ingredients for which there are data.

Additionally, the program uses several "helper" functions to perform its calculations. An example of such a function is RemoveNaN.m, which goes through Excel data that has been read in and deletes all NaN entries before the data is processed any further. Other helper functions include but are not limited to LoadList.m, which reads in lists from Excel and converts them into

1-D MATLAB structure arrays, and LoadEquipment.m, which reads in data on processing equipment and loads it into 2-D structure arrays.

Finally, the program contains two more functions for the project's "optional" functionalities:

1. **CropGrowthArea.m**

   This function reads in a list of ingredients and desired growth rates for each ingredient, and calculates the total surface area required to grow the list of ingredients. (Relies on helper function PreviousTier.m, which allows the program to trace ingredients back to their crop roots, as explained in section IV-C.)

2. **SensitivityAnalysis.m**

   This function takes in 3 crops or ingredients (currently hardcoded as potato, brown rice, and tempeh) and 3 mission parameters (currently hardcoded as power, water, and crew time) and calculates the change in ESM for each ingredient when the ESM cost of each of the 3 parameters is increased by 20%. (Relies on helper functions SensitivityAnalysis_CropESM.m, SensitivityAnalysis_CropESMLoop.m, SensitivityAnalysis_ProcessedESM.m, and SensitivityAnalysis_ProcessedESMLoop.m, which are modifications of the main crop and ingredient ESM calculation functions that allow insertion of altered mission parameters.)

B. CROP ESM CALCULATIONS

Since all ingredients considered in the analysis are either crops or the product of crops, the first calculation that must be performed is the ESM of all crops that are being considered. This calculation requires data on each crop, which is stored in Excel using the format shown below in Table II.

Table II: Sample crop data for potatoes

Attribute	Quantity	Units
harvest_index	0.7	N/A
height	0.8	m
par_intensity	725	mmol/m2
lamp_changes_per_mission	1	N/A
light_hours_per_day	12	N/A
water_cost	0	kgESM/m2
lamp_mass	0.85	kg
use_rate	0.023	kg db/p-day
plant_productivity	0.035	kg/m2-day

This data includes all parameters required to calculate ESM for the crop. Thus, CropESMLoop.m and CalculateCropESM.m can use this set of parameters for each crop to calculate all desired crop ESM values. The results of these calculations are shown below in Table III.

Table III: Crop ESM values

Crop	ESM
cabbage	7.93
carrot	7.13
chard	14.23
drybean	12.83
lettuce	9.46
pakchoi	6.19
peanut	25.68
potato	3.29
rice	14.53
soybean	18.34
spinach	10.83
sweetpotato	4.36
tomato	5.24
wheat	7.53

Calculations are done using the general ESM equation, with the main concerns being the amount of lighting needed to grow the crop (and the power and cooling requirements associated with this lighting system), how much crew time will be required to maintain this crop growth and lighting setup, how quickly the crop grows, how much it weighs, and how much of the habitat's volume it occupies. Note that all crops currently have a water cost of 0, resulting in a relatively low ESM value. These values of 0 are placeholders, and will be replaced with better estimates in a future version of the program. (For more details on how crop ESM calculations were performed, see the code for CalculateCropESM.m included in the appendix to this paper.)

The units for all ESM calculation results are kilograms of ESM per kilogram of edible dry mass. From the results obtained, it is clear that the cheapest crops on the list are potatoes, while the most expensive crops are peanuts.

C. PROCESSED INGREDIENT ESM CALCULATIONS

i. BASIC CALCULATION

Calculations for processed ingredient ESMs follow a similar format to crop ESM calculations, but cannot be performed until crop ESM values have been calculate since process ESM values rely on the ESM values of their crop inputs. For example, threshed soybean, which is the result of the threshing soybeans process, will have an ESM cost equal to that of soybeans plus the ESM costs associated with threshing them and the ESM cost of the waste produced by the thresing process, divided by the yield for the threshing process (i.e. for each kilogram of soybean processed, the yield is equal to the number of kilograms of threshed soybeans obtained).

Thus, the basic calculation for processed ingredients is as follows:

$$ESM(ingredient) = \frac{\sum ESM(inputs) + ESM(process) + ESM(waste)}{Process\ Yield}$$

These calculations require data on each process, which is stored in Excel using the format shown below in Table IV. (Note that the required data includes both the equipment table and the input and output data, which are shown as two separate tables in this paper for readability.)

Table IV: Sample data for tofu process

Equipment	Time under power, min	Instantaneous power, W	Crew labor, min	Instantaneous cooling, W	Water, kg/batch	Resupply, kg/batch
analytical_balance	5	10	10	10	0	0
soy_cow	32	3000	5	3000	0	0
manual	10	0	10	0	0	0
tupperware	15	0	15	0	0	0

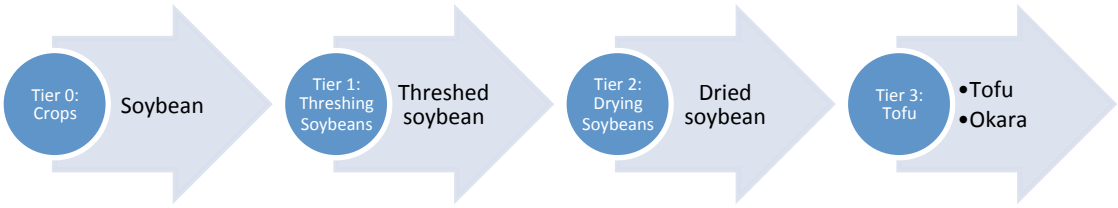
Inputs	Input Ratio	Outputs	Output Ratio
dried_soybean	1	tofu	0.8
coagulant		okara	0.2

The ESM calculations performed using this data follow the general ESM equation, but also include an extra parameter for resupply, as some processes require inputs that cannot be grown in space or require equipment that must be maintained using spare parts shipped in from Earth. The coagulant used to make tofu (shown in red in Table IV) is an example of an input that would be shipped in via resupply. Coagulant is shown in red, however, because it has not yet been incorporated into the analysis, but is a planned feature for a future version of the program. (For more details on how processed ingredient ESM calculations were performed, see the code for CalculateProcessedESM.m included in the appendix to this paper.)

ii. INGREDIENT TIERS

Some ingredients, such as tofu, are the result of multiple food processing steps, which are hereafter referred to as the ingredient's tier. Since crops are foods that result from 0 food processing steps, they are considered tier 0 ingredients. Ingredients that are the result of one food process are tier 1; ingredients that result from two processes are tier 2, and so on. The tier of an ingredient is equal to the number of processes that must be executed in order to create it. A flow chart of the ingredient tiers leading to the creation of tofu and okara (tier 3 ingredients) is shown below in Figure II.

Figure II: Sample tier breakdown for tofu process



Since the ESM of ingredients produced by each tier of processes depends on the ESM of ingredients or crops from the previous tier, ESM values must be calculated in ascending order by tier. For example, for tofu and okara, since the input for the tofu process is dried soybeans, the ESM value for dried soybean must be calculated, which relies on the ESM value for threshed soybean, which relies on the ESM value for soybean. Based on this logic, it is clear that the ESM of any ingredient ultimately relies on the ESM of the crops from which it is made. (See code for ProcessedESMLoop.m in the appendix to this paper for more details on how tiered ingredient calculations were performed for this analysis.)

Preliminary results of processed ingredient ESM calculations are shown below in Table V.

Table V: Processed ingredient ESMs

Ingredient	ESM
dried_wheat	8.05
dried_rice	14.87
dried_peanut	26.32
threshed_soybean	18.89
sweetpotato_flour	6.18
dried_soybean	19.42
wheat_berries	8.57
brown_rice	15.37
shelled_peanut	26.58
roasted_peanut	26.64
shelled_peanut_roasted	26.92
whole_wheat_flour	9.23
white_flour	13.16
bulgur_wheat	8.84
rolled_wheat_flakes	9.15
soymilk	25.43
tofu	22.77
okara	5.69
soynut	19.98
tempeh	27.09
soybean_sprouts	21.63
soy_flour	22.25
mushroom_spawn	8.67
seitan	5.12
dried_wheat_starch	3.84
dried_bran	3.84
whole_wheat_pasta	10.09
wheat_cereal	9.93
peanutbutter	27.37
peanutoil	93.21
glucose_syrup	44.42

From these results, it is evident that the cheapest ingredients in the model are low tier ingredients with cheap crop inputs (such as sweet potato flour) and ingredients that are the product of processes with many outputs (such as seitan), as well as ingredients that are by-products of more expensive ingredients (such as okara, which is produced by the tofu process, and dried bran, which is produced when making seitan). In contrast, the most expensive ingredients are high tier ingredients with expensive crop inputs (such as peanut butter).

V. SENSITIVITY ANALYSIS

Sensitivity analysis is a process that aims to describe how much a given program output is affected by a change in one of the input parameters. [7] The process is useful in situations where the exact input values are uncertain and it would be useful to know how much an error in these values would affect the final values calculated by the program. One such example is with the overall ESM calculations for crops and ingredients, since all of these calculations rely on ESM equivalency values taken from sources such as NASA's ISS BVAD. Thus, performing sensitivity analysis on selected crops and ingredients for crucial equivalency values can help elucidate the relative importance of each equivalency value to the ESM cost of food.

For the purposes of this paper and project, potato, brown rice, and tempeh have been selected as the crops and ingredients, and power, water, and crew time have been selected as the equivalency values of interest. The MATLAB script written for sensitivity analysis (SensitivityAnalysis.m) calculates the percent change in ESM for the selected crops and ingredients when power, water, and crew time are made 20% more expensive. The results of the preliminary sensitivity analysis are shown below in Table VI. (Results of water sensitivity analysis are not shown, for reasons explained below.)

Table VI: Results of sensitivity analysis

Ingredient	Initial ESM	ESM with new power value	% change in ESM	ESM with new crew time value	% change in ESM
potato	3.29	3.44	+4.84%	3.29	+0.04%
brown_rice	15.37	16.16	+5.13%	15.49	+0.77%
tempeh	27.08	28.37	+4.75%	27.41	+1.20%

The results of the sensitivity analysis show that crop and ingredient ESMs are most sensitive to changes in the cost of power, and only marginally sensitive to changes in the cost of crew time. The results also showed that crop and ingredient ESMs were in most cases not sensitive at all (0% change in ESM) to changes in the cost of water; however, this is because the water cost for many crops has been set to 0 in this version of the program, as previously mentioned. Since these 0 values are most likely not accurate, the results of the water sensitivity analysis are surely inaccurate as well, and will need to be recalculated once the proper water cost values for crops have been obtained. (For more details on how sensitivity analysis values have been calculated, see code for SensitivityAnalysis.m included in the appendix to this paper.)

The crew time result is significant for Mars mission planning, as it indicates that cooking on Mars will probably be well worth crew members' time. The complexity of a recipe and preparation time associated with cooking a given meal will be less of a concern than the ESM cost of the ingredients used to make it.

VI. CROP SURFACE AREA CALCULATIONS

In addition to the ESM of crops and ingredients, a parameter of interest for Mars mission planning is the surface area required to grow the desired crops and ingredients for a mission. Whereas previous crop surface area calculations rely on a crop use rate commensurate with a

straw man diet for each crew member, it is useful to be able to calculate the surface area required to grow a given menu or list of ingredients at a specified rate. The functionality to perform such calculations has been included in the MATLAB software package (in the file CropGrowthArea.m).

To perform surface area calculations, the program reads in a list of crops or ingredients, along with the desired quantity per week of each ingredient. Ingredients are then traced back to their crop roots, and the amount of crop needed to obtain the desired quantity of each ingredient is calculated using the yields for each tiered process used to create the ingredient. The quantities of each crop are then converted into a required surface area using the crop's plant productivity value, and finally, the required surface areas for each crop or ingredient on the list are summed up to obtain the total required surface area for the list. (For more details on how these calculations were performed, see code for CropGrowthArea.m in the appendix to this paper.)

An example list of ingredients and quantities, along with the required surface areas for each ingredient, is shown below in Table VII. (Note that the quantities are expressed in kilograms of edible dry mass.)

Table VII: Sample list of ingredients

Ingredient	Quantity (kg) per person per week	Required crop area for 6 crew members (m^2)
peanutbutter	0.5	96.30
whole_wheat_flour	2	28.60
whole_wheat_pasta	1	14.44
seitan	1	44.69
pakchoi	1	59.94
potato	1	24.49

The total surface area required to grow this list of crops and ingredients is approximately 268.46 m^2.

The results of this calculation show that expensive ingredients, such as peanut butter, require an extremely large surface area to be produced in large quantities throughout the mission. The implications of having a large habitat surface area remain uncertain: While the volume occupied by the crops and ingredients are included in their ESM calculations, other factors that have not yet been incorporated into the program include the effect of a large habitat size on crew time requirements for habitat maintenance. Although this effect has not yet been quantified, it is conceivable that increasing habitat size would increase the time needed to maintain it, especially in contingency scenarios such as storms. It thus follows that minimizing habitat size could be a desired feature in future menu and diet optimizations; however, more analysis is necessary to reach a definitive conclusion on this issue.

VII. FORWARD WORK

While the results obtained from the program thus far provide valuable insights into the cost of growing and processing food in space, several areas for improvement have been identified to increase the fidelity of the simulation. It was noted that the current model does not account for the ESM cost of storage space used to store surplus crops, nor does it account for the waste costs associated with producing ingredients that are never eaten during the mission. (For example, if the okara produced by the tofu process is never eaten, then the ESM cost of tofu should consider okara as waste.) Although these improvements have not yet been incorporated into the program, the following sections provide some discussion of possible paths forward, which may be implemented in future versions of the program.

A. STORAGE

When calculating how much of a crop is needed during the mission, the program should take into account the maximum amount of the crop that must be stored at any given time, and include the cost of this storage space as part of the crop's ESM. For example, if wheat takes 4 weeks to grow, and the crew require 1 kilogram of dry edible wheat mass for consumption each week, this means 4 kilograms of wheat must be grown for each 4 week harvest cycle. However, since only 1 kilogram is consumed each week, this means space must be allocated within the habitat to store at least 3 kilograms of dried wheat at a time. This space will have an ESM cost, as it occupies volume within the habitat. (Note that the crop must be dried before it is stored so that it can keep for 3 weeks.)

One possible solution to this issue would be to calculate the maximum surplus amount of each crop using the logic described above (i.e. calculate the use rate per harvest cycle, and use this number to determine the maximum amount of crop that must be stored at any given time), and then multiply this amount in kilograms by a volumetric packaging factor and a mass-to-volume conversion factor for the given crop. The packaging factor should be available from the BVAD or other published NASA research, while mass-to-volume conversion factors could be obtained from agricultural or culinary databases such as the United States Department of Agriculture (USDA) Nutrient Database. [8]

Thus, the calculation for ESM due to storage would be as follows:

$$ESM(storage) = \max(surplus\ of\ crop) * volumetric\ packaging\ factor * V_{eq}$$

For processed ingredients from higher tiers that require storage, the same calculation can be used, bearing in mind that increases in ESM costs due to storage will be inherited by each

subsequent tier. As a result, implementing these calculations is likely to have a dramatic effect on the ESM cost of higher tier ingredients, which rely on multiple tiers of ingredients that must be both stored and processed.

B. WASTE COSTS

Another potential ESM cost that has not been considered at this stage is the waste cost of producing ingredients that are never consumed. An example of such a scenario would be if crewmembers produce tofu for their diets, which creates a by-product of okara that the crew choose not to consume. In such a scenario, the okara should be considered waste, and should be given an ESM cost commensurate with the typical waste costs for the mission.

As it would be impossible to know which ingredients would be wasted until a menu or diet optimization has been performed, these corrections would need to take place once an optimized menu or diet has been created, resulting in an iterative optimization process that continues until the waste costs do not make the selected diet more expensive than other alternatives. The new waste calculation could then be performed by limiting the yield of a given process to outputs that are actually used during the mission, and classifying all other outputs as waste.

For example, the tofu process has a yield of 0.694 and outputs tofu and okara in ratios of 0.8 and 0.2, respectively. Failure to consume the okara would thus mean that the process has a yield of 0.555, and has tofu as its only output with a ratio of 1.

C. PROCESSING CALCULATIONS

An improvement that could be made to the current processed ingredient ESM calculations would be to calculate certain equipment data values, such as powered time, by taking a multiple of the process's batch size instead of hardcoding the values into Excel.

For example, the time under power required for threshing processes is always dependent on the batch size. For threshing soybeans, the process currently calls for 38.57 minutes of time under power for the bean thresher, and this value is hardcoded into Excel. However, the number 38.57 is obtained by multiplying the batch size (currently calculated as approximately 9.64) by 4. Therefore, a desirable improvement to the program structure would be to allow users to input the factor by which batch size should be multiplied into the spreadsheet, rather than the hardcoded value.

A possible method of implementation would be to ask users for 3 numbers instead of 1, as shown below in Table VIII. These 3 numbers could be used to perform equipment usage calculations using a slope-intercept equation.

Table VIII: Example processing data for threshing process

Equipment	Time under power, min	Instantaneous power, W	Crew labor, min	Instantaneous cooling, W	Water, kg/batch	Resupply, kg/batch
bean_thresher	0,4,2	0,0,4000	0,0,5	0,0,4000	0,0,0	0,0,0

In the cells shown above, the first number is a Boolean that indicates whether or not the calculation shall be done using a step function (in which case a ceiling function would be used), the second number represents the slope of our equipment usage line, and the third number represents its intercept. Therefore, the time under power for the bean thresher would be equal to $4x + 2$, where x is the batch size. (The intercept value of 2 minutes represents the time needed to

set up the machine before running it and to discharge the last few beans remaining at the end of the batch.)

Another concern with the current method processing calculations is the weight per usage for each piece of equipment. Currently, calculations are performed using placeholder guesses for weight per usage. Once menu and diet optimizations can be performed, however, the number of times each piece of equipment is used throughout the mission will become a known quantity, and this quantity can be used to calculate more precise values for weight per usage. (As with the new procedure for waste cost calculations described in section VII-B, such calculations would result in an iterative optimization process that would continue until the weight of the equipment does not make the selected diet more expensive than other alternatives.)

E. WATER COSTS

As mentioned in previous sections, current data spreadsheets for crops currently list 0 as the value for water costs, which is inaccurate. Future versions of the program should include more realistic water costs, either as hardcoded values or as values calculated using other crop and mission parameters.

VIII. CONCLUSIONS

This paper and project lay the initial groundwork for future menu and diet optimizations for long-duration space missions using ESM. While limitations on available menu data currently pose obstacles towards performing the full optimization, it is anticipated that once those data are obtained, the MATLAB program and the calculation methods it includes for crop and ingredient ESMs will form the basis of menu optimization efforts.

As discussed in section V, the results of this preliminary analysis show that the ESM cost of preparing food in space is sufficiently low that the concept of growing, processing, and cooking food in deep space merits further investigation. Future versions of this software that incorporate suggestions from this paper's "Forward Work" section (VII) should provide reasonably accurate figures for crop and ingredient ESMs, and should also allow program users to determine which ingredients should be shipped in from Earth instead of being processed in space. Current results from the program suggest that ingredients with extremely high ESMs, such as peanut butter, should either be avoided on missions or shipped in on resupply missions for efficiency's sake.

Also important to note are the limitations of simulation, as the calculations and planned optimization do not take into account crewmember preferences. Once menu data are obtained and diets with attractive ESM costs are identified, menus should also be tested with human subjects to confirm that they are indeed palatable to crewmembers, and are not just attractive on paper. Possible venues for menu testing include the Hawai'i Space Exploration Analog and Simulation (HI-SEAS) and NASA's Human Exploration Research Analog (HERA). [9]

Finally, it should be noted that the work described in this paper concerns only surface missions to Mars, and does not take into account the journeys between Mars and Earth. Since this travel time will likely be in a completely closed loop spacecraft in zero gravity, standard food processing techniques will not apply, and alternate solutions will need to be explored. It is recommended that researchers begin exploring the possibility of food growth and processing in zero gravity in the near future, as manned missions to Mars will likely require the use of novel processing techniques to maintain an adequate supply of food during flight.

IX. REFERENCES

[1] "NASA's Journey to Mars." *National Aeronautics and Space Administration*, https://www.nasa.gov/content/nasas-journey-to-mars. Accessed 18 December 2016.

[2] Hoffman, S. J. and Kaplan, D. I. "Exploration of Mars: The Reference Mission of the NASA Mars Exploration Study Team." *NASA Center for Aerospace Information*, 1997.

[3] Levri, J. A. et al. "Advanced Life Support System Mass Guidelines Document", 2003. NASA/TM-2003-212278, A-0310698. Presented at Purdue University, 13 Mar. 2003. Available at http://ntrs.nasa.gov/search.jsp?R=20040021355.

[4] "Ideal Rocket Equation." *National Aeronautics and Space Administration*, https://spaceflightsystems.grc.nasa.gov/education/rocket/rktpow.html. Accessed 18 December 2016.

[5] Jones, H. "Equivalent Mass versus Life Cycle Cost for Life Support Technology Selection", 2003. SAE Technical Paper 2003-01-2635. Presented at 33rd International Conference on Environmental Systems, July 2003, Vancouver, BC, Canada.

[6] Anderson, M. et al. "Life Support Baseline Values and Assumptions Document", March 2015. NASA TP-2015-218570. Available at http://ston.jsc.nasa.gov/collections/TRS/.

[7] Loucks, D. P. and van Beek, E. *Water Resources Systems Planning and Management: An Introduction to Methods, Models and Applications*. United Nations Educational, Scientific and Cultural Organization, 2005.

[8] "USDA Food Composition Databases." *United States Department of Agriculture*, https://ndb.nal.usda.gov/ndb/. Accessed 18 December 2016.

[9] Lieberman, J. "Mars Food Researchers Emerge from HI-SEAS Dome After 4 Month Cooking Experiment." *International Science Times*, 14 August 2013. Accessed 18 December 2016.

X. APPENDIX: SELECTED MATLAB CODE

A. CalculateCropESM.m

```
% This function reads in the name of a crop and then performs all necessary
% calculations to compute the crop's ESM.

function CropESM = CalculateCropESM(crop)

% Read in spreadsheets containing infrastructure, lighting, crop and mission
% data.
[infrastructureNum,infrastructureTxt,infrastructureRaw] =
xlsread('Infrastructure.xlsx');
[lightingNum,lightingTxt,lightingRaw] = xlsread('Lighting.xlsx');
[missionNum,missionTxt,missionRaw] = xlsread('MissionData.xlsx');

% % % % % % % % % %

% Load variables that are common to all crops.

% Infrastructure Parameters:
infrastructure = struct();
infrastructure_names = infrastructureTxt(:,1);

for j = 1:length(infrastructureNum)
    infrastructure.parameter(j) = infrastructure_names(j+2);

    % Check to make sure all the values are positive numbers.
    if infrastructureNum(j) < 0
        s1 = 'Error: ';
        s2 = infrastructure.parameter(j).name;
        s3 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(s1,s2),s3);
    end

    % If numbers check out, assign the values to the corresponding
    % parameters.
    infrastructure.value(j) = infrastructureNum(j);
end

% Mission Parameters:
mission = struct();
mission_variables = missionTxt(:,1);

for j = 1:length(missionNum)
    mission.parameter(j) = mission_variables(j+2);

    % Check to make sure all the values are positive numbers.
    if missionNum(j) < 0
        s1 = 'Error: ';
        s2 = mission.parameter(j).name;
```

```
        s3 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(s1,s2),s3);
    end

    % If numbers check out, assign the values to the corresponding
    % parameters.
    mission.value(j) = missionNum(j);
end

% Lighting Parameters:
lighting = struct();
lighting_variables = lightingTxt(:,1);

for j = 1:length(lightingNum)
    lighting.parameter(j) = lighting_variables(j+2);

    % Check to make sure all the values are positive numbers.
    if lightingNum(j) < 0
        s1 = 'Error: ';
        s2 = lighting.parameter(j).name;
        s3 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(s1,s2),s3);
    end

    % If numbers check out, assign the values to the corresponding
    % parameters.
    lighting.value(j) = lightingNum(j);
end

% Now read in the name of a crop to obtain crop-specific data.

fileExtension = '.xlsx';
fileFolder = 'Crops/';
fileName = strcat(crop,fileExtension);
filePointer = strcat(fileFolder,fileName);

% First, read in the appropriate spreadsheet.
[cropNum,cropTxt,cropRaw] = xlsread(filePointer);

% Now load all of our crop variables into a structure array for use in
% ESM calculations.
crop = struct();
crop_parameters = cropTxt(:,1);

for j = 1:length(cropNum)
    crop.parameter(j) = crop_parameters(j+2);

    % Check to make sure the values are all positive numbers.
    if cropNum(j) < 0
        s1 = 'Error: ';
        s2 = crop.parameter(j).name;
        s3 = ' for crop ';
        s4 = crop;
        s5 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
```

```
    end

    % If number checks out, assign the value to the corresponding
    % parameter.
    crop.value(j) = cropNum(j);
end

% Assign meaningful variable names to all crop parameters.
for j = 1:length(crop.parameter)
    if strcmp(crop.parameter(j),'harvest_index')
        crop_harvest_index = crop.value(j);
    elseif strcmp(crop.parameter(j),'height')
        crop_height = crop.value(j);
    elseif strcmp(crop.parameter(j),'par_intensity')
        crop_par_intensity = crop.value(j);
    elseif strcmp(crop.parameter(j),'lamp_changes_per_mission')
        crop_lamp_changes_per_mission = crop.value(j);
    elseif strcmp(crop.parameter(j),'light_hours_per_day')
        crop_light_hours_per_day = crop.value(j);
    elseif strcmp(crop.parameter(j),'water_cost')
        crop_water_cost = crop.value(j);
    elseif strcmp(crop.parameter(j),'lamp_mass')
        crop_lamp_mass = crop.value(j);
    elseif strcmp(crop.parameter(j),'use_rate')
        crop_use_rate = crop.value(j);
    elseif strcmp(crop.parameter(j),'plant_productivity')
        plant_productivity = crop.value(j);
    end
end

% Calculate crop area per person using use rate and plant productivity.
crop_area_per_person = crop_use_rate/plant_productivity;

% Calculate the lighting ESM. (To do this, we need to search within our
% previously loaded data to find all the variables we are using.)

% First, search for our infrastructure variables.
for j = 1:length(infrastructure.parameter)
    if strcmp(infrastructure.parameter(j),'lamps_per_m2')
        lamps_per_m2 = infrastructure.value(j);
    elseif strcmp(infrastructure.parameter(j),'pressurized_volume')
        pressurized_volume = infrastructure.value(j);
    elseif strcmp(infrastructure.parameter(j),'heat_rejection')
        heat_rejection = infrastructure.value(j);
    elseif strcmp(infrastructure.parameter(j),'power')
        power = infrastructure.value(j);
    elseif strcmp(infrastructure.parameter(j),'crew_time')
        crew_time = infrastructure.value(j);
    elseif strcmp(infrastructure.parameter(j),'other_production_costs')
        other_production_costs = infrastructure.value(j);
    elseif strcmp(infrastructure.parameter(j),'nds_equipment_mass')
        nds_equipment_mass = infrastructure.value(j);
    elseif strcmp(infrastructure.parameter(j),'waste_costs')
        waste_costs = infrastructure.value(j);
    end
end
```

```
% Now search for mission-specific variables.
for j = 1:length(mission.parameter)
    if strcmp(mission.parameter(j),'duration')
        mission_duration = mission.value(j);
    end
end

% Calculate total kg per person
total_kg_p = crop_use_rate * mission_duration;

% Finally, search for lighting-specific variables.
for j = 1:length(lighting.parameter)
    if strcmp(lighting.parameter(j),'lamp_power')
        lamp_power = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'lamp_mass')
        lamp_mass = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'ballast')
        ballast = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'ballast_power')
        ballast_power = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'coldplate_etc')
        coldplate_etc = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'height_lighting_assembly')
        height_lighting = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'lamp_volume')
        lamp_volume = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'lamp_change_crewtime')
        lamp_change_crewtime = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'changes_per_mission')
        changes_per_mission = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'lamp_resupply_mass_factor')
        lamp_resupply_mass_factor = lighting.value(j);
    elseif strcmp(lighting.parameter(j),'lamp_resupply_volume_factor')
        lamp_resupply_volume_factor = lighting.value(j);
    end
end

cropLightingESM_lamp = lamp_mass * lamps_per_m2 * (crop_par_intensity/1000);
cropLightingESM_ballast = ballast * lamps_per_m2 * (crop_par_intensity/1000);
cropLightingESM_etc = coldplate_etc;
cropLightingESM_kg = cropLightingESM_lamp + cropLightingESM_ballast +
cropLightingESM_etc;
cropLightingESM_m3 = height_lighting/pressurized_volume;

% ESM per kW power and cooling requires calculation of sub-components:
cropLightingESMkWp_lamp = (lamp_power*lamps_per_m2*crop_par_intensity)/1000;
cropLightingESMkWp_ballast =
(ballast_power*lamps_per_m2*crop_par_intensity)/1000;
cropLightingESMkWp = (cropLightingESMkWp_lamp +
cropLightingESMkWp_ballast)*1000*((crop_light_hours_per_day/24)/power);
cropLightingESMkWc = (cropLightingESMkWp_lamp +
cropLightingESMkWp_ballast)*1000*((crop_light_hours_per_day/24)/heat_rejectio
n);

% Calculate lamp mh/y to find overall mh/y:
```

```
cropLightingESMmhy_lamp =
crop_lamp_changes_per_mission*lamps_per_m2*lamp_change_crewtime;
cropLightingESMmhy =
(cropLightingESMmhy_lamp*(mission_duration/365))/crew_time;

% Calculate kg/y and m3/y:
cropLightingESMkgy = changes_per_mission * (crop_lamp_mass/mission_duration)
* lamp_resupply_mass_factor;
cropLightingESMm3y = lamp_volume * crop_lamp_changes_per_mission *
lamps_per_m2 * (crop_par_intensity/1000)/mission_duration *
lamp_resupply_volume_factor;

% Add all previous calculations up to obtain the total lighting ESM:
cropLightingESM = cropLightingESM_kg + cropLightingESM_m3 +
cropLightingESMkWp + cropLightingESMkWc + cropLightingESMmhy +
cropLightingESMkgy + cropLightingESMm3y;

% Calculate the lighting investment per square meter:
cropLightingInvestment = cropLightingESM * crop_area_per_person;

% Calculate the water, laber and NDS per unit area:
cropUnitArea = (crop_water_cost + other_production_costs +
nds_equipment_mass) * crop_area_per_person;

% Calculate the pressurized volume devoted to the plants themselves. (This
% term drops out according to the values in the current Excel spreadsheet.)

% Calculate the cost to process agricultural waste from the crop:
cropAgWaste = total_kg_p * waste_costs *((1-
crop_harvest_index)/crop_harvest_index);

% Now add all 4 terms together to calculate the total mission ESM:
MissionESM = cropLightingInvestment + cropUnitArea + cropAgWaste;

% Divide by the total use rate for the mission to obtain kg ESM/kg:
CropESM = MissionESM/(crop_use_rate*mission_duration);

end
```

B. CalculateProcessedESM.m

```
function ProcessESM = CalculateProcessedESM(process_name,masterESMlist)

% Read in data that is common to all processes. This includes equipment
% specs, processing data, mission data, and infrastructure data.
[equipmentSpecNum,equipmentSpecTxt,equipmentSpecRaw] =
xlsread('Processing/equipment_specs.xlsx');
[dataNum,dataTxt,dataRaw] = xlsread('Processing/processing_data.xlsx');
[missionNum,missionTxt,missionRaw] = xlsread('MissionData.xlsx');
[infrastructureNum,infrastructureTxt,infrastructureRaw] =
xlsread('Infrastructure.xlsx');

% Load mission data for calculations by searching within our mission data
% for our desired parameters.
for j = 1:length(missionNum)
    if strcmp(missionTxt(j+2,1),'number_of_crew')
        mission_crew = missionNum(j);
    elseif strcmp(missionTxt(j+2,1),'duration')
        mission_duration_days = missionNum(j);
    elseif strcmp(missionTxt(j+2,1),'diet_total')
        diet_total = missionNum(j);
    end
end

% Calculate the mission duration in Mars minutes using the number of days:
mission_duration_marsminutes = mission_duration_days * 1440;

% Load infrastructure data for processing ESM calculations by searching
% within our infrastructure data for our desired parameters and assigning
% each a variable name.
for j = 1:length(infrastructureNum)
    if strcmp(infrastructureTxt(j+2,1),'power')
        infrastructure_power = infrastructureNum(j);
    elseif strcmp(infrastructureTxt(j+2,1),'heat_rejection')
        infrastructure_heatrejection = infrastructureNum(j);
    elseif strcmp(infrastructureTxt(j+2,1),'crew_time')
        infrastructure_crewtime = infrastructureNum(j);
    elseif strcmp(infrastructureTxt(j+2,1),'reprocessed_water')
        infrastructure_water = infrastructureNum(j);
    elseif strcmp(infrastructureTxt(j+2,1),'mass_delivery_factor')
        infrastructure_massdelivery = infrastructureNum(j);
    elseif strcmp(infrastructureTxt(j+2,1),'volume_delivery_factor')
        infrastructure_volumedelivery = infrastructureNum(j);
    elseif strcmp(infrastructureTxt(j+2,1),'packaging_factor')
        infrastructure_packaging = infrastructureNum(j);
    end
end

% Load data common to all processes (name, interval, yield, batch size,
% batches per mission).
process_names = dataRaw(:,1);
process_intervals = dataNum(:,1);
process_yields = dataNum(:,2);
```

```
% Search within processing data to find key number for desired process.

% Set initial value of process key to 0 (real value must be great than or
% equal to 1). Create placeholder string to hold string values of process
% names.
process_key = 0;

% Loop through all process names until we find our desired process.
process_names_length = length(process_names);

for j = 1:process_names_length
    cellContents = char(process_names{j,1});
    if strcmp(cellContents,process_name)
        process_key = j;
    end
end

% Check to make sure that there are no errors in the process name. If the
% process name does not match any of the entries on our master process
% list, display an error message.
if process_key == 0
    s1 = 'Error: Process ';
    s2 = process_name;
    s3 = ' is not a valid process.';
    errorMsg = strcat(strcat(s1,s2),s3);
    error(errorMsg)
end

% If process name is valid, load data for specific process.
fileExtension = '.xlsx';
fileFolder = 'Processing/';
fileName = strcat(process_name,fileExtension);
filePointer = strcat(fileFolder,fileName);

[equipmentNum,equipmentTxt,equipmentRaw] = xlsread(filePointer);

% Using the identified process key, load values for interval and yield.
% Check to make sure that the interval and yield are both positive numbers,
% and throw an error message if they are not.

field1 = 'interval';

if process_intervals(process_key-1) <= 0
    s1 = 'Error: Process interval for ';
    s2 = process_name;
    s3 = ' must be a number greater than 0.';
    errorMsg = strcat(strcat(s1,s2),s3);
    error(errorMsg)
else
    interval_value = process_intervals(process_key-1);
end

field2 = 'yield';
```

```
if process_yields(process_key-1) <= 0
    s1 = 'Error: Process yield for ';
    s2 = process_name;
    s3 = ' must be a number greater than 0.';
    errorMsg = strcat(strcat(s1,s2),s3);
    error(errorMsg)
else
    yield_value = process_yields(process_key-1);
end

% Create structure array to store all data for process.
process = struct(field1,interval_value,field2,yield_value);

% Calculate batches per mission and batch size from previously loaded data
% and add this to the structure array.
batches_per_mission_value = mission_duration_days / process.interval;
[process(:).batches_per_mission] = batches_per_mission_value;

batch_size_value = (diet_total*mission_crew*process.interval)/7000;
[process(:).batch_size] = batch_size_value;

process.tier = dataNum(process_key-1,3);

% Load process inputs and get rid of NaN entries.
[process(:).inputs] = LoadList(equipmentRaw(:,8));
process.inputs = RemoveNaN(process.inputs);

% Load process input ratios.
[process(:).input_ratios] = LoadList(equipmentNum(:,8));

% Check to make sure input ratios are all between 0 and 1 and add up to 1.
input_ratios_total = 0;

for j = 1:length(process.input_ratios)
    marginal_input_ratio = process.input_ratios(j);
    if isnan(marginal_input_ratio) == false
        if marginal_input_ratio <= 0
            s1 = 'Error: Input ratio for ';
            s2 = process.inputs{1,j+1};
            s3 = ' in process ';
            s4 = process_name;
            s5 = ' must be a number greater than 0.';
            errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
            error(errorMsg)
        elseif marginal_input_ratio > 1
            s1 = 'Error: Input ratio for: ';
            s2 = process.inputs{1,j+1};
            s3 = ' in process: ';
            s4 = process_name;
            s5 = ' cannot be greater than 1.';
            errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
            error(errorMsg)
        end
        input_ratios_total = input_ratios_total + marginal_input_ratio;
```

```
    end
end

if input_ratios_total ~= 1
    s1 = 'Error: Input ratios for process: ';
    s2 = process_name;
    s3 = ' do not add up to 1.';
    errorMsg = strcat(strcat(s1,s2),s3);
    error(errorMsg)
end

% Load process equipment and equipment data.
[process(:).equipment] = LoadEquipment(equipmentRaw,equipmentNum);

% Define process output and get rid of NaN entries.
[process(:).outputs] = LoadList(equipmentRaw(:,10));
process.outputs = RemoveNaN(process.outputs);

% Load process output ratios and get rid of NaN entries.
[process(:).output_ratios] = LoadList(equipmentNum(:,10));

% Check to make sure output ratios are all between 0 and 1, and that they
% collectively add up to 1.
output_ratios_total = 0;

for j = 1:length(process.output_ratios)
    marginal_output_ratio = process.output_ratios(j);
    if isnan(marginal_output_ratio) == false
        if marginal_output_ratio <= 0
            s1 = 'Error: Output ratio for ';
            s2 = process.outputs{1,j+1};
            s3 = ' in process ';
            s4 = process_name;
            s5 = ' must be a number greater than 0.';
            errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
            error(errorMsg)
        elseif marginal_output_ratio > 1
            s1 = 'Error: Output ratio for: ';
            s2 = process.outputs{1,j+1};
            s3 = ' in process: ';
            s4 = process_name;
            s5 = ' cannot be greater than 1.';
            errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
            error(errorMsg)
        end
        output_ratios_total = output_ratios_total + marginal_output_ratio;
    end
end

if output_ratios_total ~= 1
    s1 = 'Error: Output ratios for process: ';
    s2 = process_name;
    s3 = ' do not add up to 1.';
    errorMsg = strcat(strcat(s1,s2),s3);
    error(errorMsg)
```

```
end

% Now that we have our data, calculate the ESM for an ingredient. First,
% we add up all the ESMs for the inputs.
numberOfInputs = length(process.inputs);
totalInputESM = 0;

% This loop goes through all the previous tiers of crops and ingredients,
% searches for each input, and adds the ESM of each input to the total
% input ESM. The loop also checks for errors in case the input does not
% match any crops or ingredients in the previous tiers.
inputESM = 0;

for j = 2:numberOfInputs
    if strcmp(process.inputs{1,j},'') == false
        for k = 1:process.tier
            for l = 1:length(masterESMlist(k).values)
                if strcmp(process.inputs(j),masterESMlist(k).values(l).name)
                    inputESM = masterESMlist(k).values(l).ESM *
process.input_ratios(j-1);
                    totalInputESM = totalInputESM + inputESM;
                end
            end
        end
        if inputESM == 0
            s1 = 'Error: Input ';
            s2 = char(process.inputs(j));
            s3 = ' is not a valid input.';
            errorMsg = strcat(strcat(s1,s2),s3);
            error(errorMsg)
        end
    end
end

% Now add the ESM for the process.
equipment_length = length(process.equipment);

% Calculate ESM for power:
powerTotal = 0;
powerESM = 0;
equipment_power = 0;

for j = 1:equipment_length

    % Check to make sure that powered time and instantaneous power are
    % positive numbers.
    if process.equipment(j).powered_time < 0
        s1 = 'Error: Powered time for ';
        s2 = process.equipment(j).name;
        s3 = ' in process ';
        s4 = process_name;
        s5 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5)
    elseif process.equipment(j).instantaneous_power < 0
        s1 = 'Error: Instantaneous power for: ';
```

```
        s2 = process.equipment(j).name;
        s3 = ' in process: ';
        s4 = process_name;
        s5 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5)
        error(errorMsg)
    end

    % Now calculate the total power for each piece of equipment and sum all
    % of them up to get the total power for the process.
    equipment_power = process.equipment(j).powered_time *
process.equipment(j).instantaneous_power;
    powerTotal = powerTotal + equipment_power;
end

% Calculate total power ESM by multiplying our power total by the number of
% times we run the process during the mission (mission duration divided by
% interval) and then dividing by the ESM cost of power for the mission.
% (Note that all of this must be calculated using Mars minutes.)
powerTotal = powerTotal/mission_duration_marsminutes;
powerESM =
(powerTotal*(mission_duration_days/process.interval))/infrastructure_power;

% Calculate ESM for labor:
laborTotal = 0;
laborESM = 0;
equipment_labor = 0;

for j = 1:equipment_length
    % Check to make sure that the labor time listed in the file is a
    % positive number.
    if process.equipment(j).crew_labor < 0
        s1 = 'Error: Crew time for ';
        s2 = process.equipment(j).name;
        s3 = ' in process '
        s4 = process_name;
        s5 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
    end

    % If number checks out, then sum up the labor time for all pieces of
    % equipment to obtain the "labor total" for the process.
    equipment_labor = process.equipment(j).crew_labor;
    laborTotal = laborTotal + equipment_labor;
end

% Multiply our labor total by the number of times the process is repeated
% over the course of the mission (mission duration divided by process
% interval) and divide by the ESM cost of crew time to get our total labor
% ESM.
laborESM =
((laborTotal/60)*(mission_duration_days/process.interval))/infrastructure_cre
wtime;

% Calculate ESM for cooling (will be similar to power ESM, but adjusted for
```

```
% the power required for heat rejection on the mission).
coolingTotal = 0;

for j = 1:equipment_length
    % Check to make sure that the labor time listed in the file is a
    % positive number.
    if process.equipment(j).instantaneous_cooling < 0
        s1 = 'Error: Instantaneous cooling power for ';
        s2 = process.equipment(j).name;
        s3 = ' in process ';
        s4 = process_name;
        s5 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
    end

    % If number checks out, multiply powered time by cooling power for each
    % piece of equipment and sum all numbers together to get the cooling
    % total for the process.
    equipment_cooling = process.equipment(j).powered_time *
process.equipment(j).instantaneous_cooling;
    coolingTotal = coolingTotal + equipment_cooling;
end

% Convert to Mars minutes and then multiply by the number of times the
% process is executed before dividing by the ESM cost of heat rejection to
% obtain the overall cooling ESM.
coolingTotal = coolingTotal/mission_duration_marsminutes;
coolingESM =
(powerTotal*(mission_duration_days/process.interval))/infrastructure_heatreje
ction;

% Calculate ESM for water:
waterTotal = 0;
waterESM = 0;
equipment_water = 0;

for j = 1:equipment_length
    % Check to make sure that the amount of water is a positive number.
    if process.equipment(j).water < 0
        s1 = 'Error: Amount of water for ';
        s2 = process.equipment(j).name;
        s3 = ' in process '
        s4 = process_name;
        s5 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
    end

    % If number checks out, simply sum up the amount of water for all
    % pieces of equipment to obtain the total amount of water for the
    % process.
    equipment_water = process.equipment(j).water;
    waterTotal = waterTotal + equipment_water;
end

% Multiply by the number of times the process is executed during the
```

```
% mission and divide by the ESM cost of water to get the total ESM cost of
% water for the process during the mission.
waterESM =
(waterTotal*(mission_duration_days/process.interval))/infrastructure_water;

% Calculate ESM for resupply:
resupplyTotal = 0;
resupplyESM = 0;
equipment_resupply = 0;

for j = 1:equipment_length
    % Check to make sure that the labor time listed in the file is a
    % positive number.
    if process.equipment(j).resupply < 0
        s1 = 'Error: Resupply cost for ';
        s2 = process.equipment(j).name;
        s3 = ' in process '
        s4 = process_name;
        s5 = ' must be a number greater than or equal to 0.';
        errorMsg = strcat(strcat(strcat(strcat(s1,s2),s3),s4),s5);
    end

    % If number checks out, sum up the resupply values for all pieces of
    % equipment to obtain the resupply total for the process.
    equipment_resupply = process.equipment(j).resupply;
    resupplyTotal = resupplyTotal + equipment_resupply;
end

% Multiply by the packaging factor, then multiply by the number of times
% the process is executed and divide by the mass delivery factor to obtain
% the total resupply ESM for the process during the mission.
resupplyESM =
((infrastructure_packaging*resupplyTotal)*(mission_duration_days/process.inte
rval))/infrastructure_massdelivery;

% Calculate ESM for equipment weight based on data in the equipment specs
% spreadsheet.
equipmentWeight = 0;
cellContents = '';

% Loop through all equipment until we find the equipment of interest, and
% then add the weight per use for that equipment to our total equipment
% weight. The loop also checks to make sure all pieces of equipment are
% valid (if the equipment does not match anything listed in our master
% equipment file, the program will throw an error).
equipment_spec_length = length(equipmentSpecRaw);

for j = 1:length(process.equipment)
    equipment_spec_key = 0;
    nameOfEquipment = char(process.equipment(j).name);
    for k = 2:equipment_spec_length
        cellContents = char(equipmentSpecRaw{k,1});
        if strcmp(cellContents,nameOfEquipment)
            equipment_spec_key = k;
```

```
            % Equipment weight per use is currently calculated using a
            % guesstimate of the number of times the equipment will be used
            % throughout the mission. This will eventually be replaced with
            % a number calculated via menu optimization.
            equipment_weight_per_use = equipmentSpecNum(equipment_spec_key-
1,1)/equipmentSpecNum(equipment_spec_key-1,2);
            equipmentWeight = equipmentWeight + equipment_weight_per_use;
        end
    end
    if equipment_spec_key == 0
        s1 = 'Error: Equipment name ';
        s2 = nameOfEquipment;
        s3 = ' is not valid.';
        errorMsg = strcat(strcat(s1,s2),s3);
        error(errorMsg)
    end
end

% Now that we have our value for total equipment weight, we can add up our
% power, labor, cooling, water, resupply and equipment weight ESMs to get
% the total ESM for the processing.
totalMissionProcessingESM = powerESM + laborESM + coolingESM + waterESM +
resupplyESM + equipmentWeight;
totalProduction_kg = process.yield * process.batches_per_mission;
processESM = totalMissionProcessingESM/totalProduction_kg;

% Now add the ESM for the waste. (We are saying that it is double its ESM
% volume cost per unit density for now, but this is a placeholder.)
waste_kg = 1 - process.yield;
wasteESM = waste_kg * 0.002/infrastructure_volumedelivery;

% Add these 3 components up to get the total ESM for the process.
total_process_ESM = (totalInputESM + processESM + wasteESM)/process.yield;

% Divide the total process ESM equally between the number of ingredients to
% get the final ESM per ingredient.
numberOfOutputs = 0;

% Loop through the output cell array and count the number of non-blank
% entries to get the total number of outputs:
for j = 2:length(process.outputs)
    if strcmp(process.outputs{1,j},'') == false
        numberOfOutputs = numberOfOutputs + 1;
    end
end

% Return structure array containing the name of each output and its
% corresponding ESM.

output = struct();

for j = 1:numberOfOutputs
    output(j).name = char(process.outputs{1,1+j});
    output(j).ESM = total_process_ESM * process.output_ratios(j);
end
```

```
ProcessESM = output;

end
```

C. CropESMLoop.m

```
function ESMLoop = CropESMLoop(input_data)

% This function takes a list of crops or ingredients and calculates the ESM
% for each one. The data is stored in a structure array and consists of
% both the name of the crop and its ESM.

% Count the number of crops or ingredients in our list. (The loop in this
% function will repeat as many times as there are ingredients.)
DataLength = length(input_data);

% Create a structure array to store our crop/ingredient names and ESM
% calculations.
ESMLoop = struct();

for j = 1:(DataLength)
    % Convert each item from input data sheet into a string
    ingredientName = char(input_data(j));

    % Add the name of the crop or ingredient to our data structure
    [ESMLoop(j).name] = ingredientName;

    % Calculate the ESM of the crop or ingredient and add it to the data
    % structure
    [ESMLoop(j).ESM] = CalculateCropESM(ingredientName);
end
```

D. ProcessedESMLoop.m

```
function ESMLoop = ProcessedESMLoop(input_data,tieredESMlist)

% This function takes a list of processes and calculates the ESM for the
% outputs of each process.

% Count the number of processes on our list. (The loop in this
% function will repeat as many times as there are processes.)
DataLength = length(input_data);

% Create a structure array to store our output names and their
% corresponding ESMs.
ESMLoop = struct();

% Create a variable to count the total number of outputs produced by
% this tier of processes.
k = 1;

for j = 1:(DataLength)
    % Read in each process from our list for the given tier.
    processName = char(input_data(j));

    % Calculate the ESM of all outputs for the process.
    outputESMs = CalculateProcessedESM(processName,tieredESMlist);

    % Now, loop through all outputs and add them to our list of ESMs for
    % this tier.

    for l = 1:length(outputESMs)
        % Add the name of the crop or ingredient to our data structure
        [ESMLoop(k).name] = outputESMs(l).name;

        % Calculate the ESM of the crop or ingredient and add it to the data
        % structure
        [ESMLoop(k).ESM] = outputESMs(l).ESM;

        % Increment k by 1 before we run through the loop again.
        k = k + 1;
    end
end

end
```

E. Main.m

```
% % % % % % % % % %

% For our main function, we read in lists for each tier of ingredients and
% then perform ESM calculations on each crop or ingredient. Calculations
% must be done in ascending order by tier number, since higher tier
% ingredient ESMs depend on lower tier calculations. The results of the ESM
% calculations are stored in structure arrays, with one array for each
% tier.

% First, read in mission data. (These variables apply to all crops and
% ingredients.)

% Read in crop and process lists.

[cropListNum,cropListTxt,cropListRaw] = xlsread('Crops/crop_list.xlsx');
[processListNum,processListTxt,processListRaw] =
xlsread('Processing/processing_data.xlsx');

% From processing data, find our maximum process tier.

maximum_tier = 0;

for j = 1:length(processListNum)
    if processListNum(j,3) > maximum_tier
        maximum_tier = processListNum(j,3);
    end
end

% Create a structure array to store processes by tier.
process_tier = struct();

% Create a counter variable for each tier (this will be used in the for
% loop that immediately follows).
for j = 1:maximum_tier
    process_tier(j).counter = 1;
end

% Populate the structure array with data from the list of processes.

for j = 1:length(processListNum)
    for k = 1:maximum_tier
        if processListNum(j,3) == k
            [process_tier(k).name(process_tier(k).counter)] =
processListRaw(j+1);
            process_tier(k).counter = process_tier(k).counter + 1;
        end
    end
end

% Create structure array to store data for crop and ingredient calculations
```

```
tieredESMlist = struct();

% Now run our ESM loop function to populate the structure array with ESMs
% for all crops.
tieredESMlist(1).values = CropESMLoop(cropListTxt);

% Run the ESM loop to populate the structure array with ESMs for all
% process outputs.
for j = 1:maximum_tier
    tieredESMlist(j+1).values =
ProcessedESMLoop(process_tier(j).name,tieredESMlist);
end

% Finally, for readability, consolidate all the ESMs we have calculated
% into one master list.
ConsolidatedData = ConsolidateESM(tieredESMlist);
```

F. SensitivityAnalysis.m

```
% This script looks at 3 key ingredients (potatoes, brown rice, and
% tempeh) and performs a simple sensitivity analysis on each one by looking
% at the effect of changing the ESM cost of power, water and labor on
% the ESM of the ingredient or crop.

% *** Note: You must run Main.m before running this file, or else your
% results either will not compile or will be incorrect. ***

% First, create a structure array to store data for each crop or
% ingredient.

analysis = struct();

analysis(1).name = 'potato';
analysis(2).name = 'brown_rice';
analysis(3).name = 'tempeh';

% Record the baseline ESM for each ingredient.
for j = 1:length(analysis)
    for k = 1:length(ConsolidatedData)
        if strcmp(ConsolidatedData(k).name,analysis(j).name)
            analysis(j).initial_ESM = ConsolidatedData(k).ESM;
        end
    end
end

% Now, for each ingredient, calculte the new ESM when we change the 3
% parameters.

% For potato:
analysis(1).power_ESM =
SensitivityAnalysis_CropESM('potato','Infrastructure_Power');
analysis(1).power_change = analysis(1).power_ESM/analysis(1).initial_ESM;
analysis(1).water_ESM =
SensitivityAnalysis_CropESM('potato','Infrastructure_Water');
analysis(1).water_change = analysis(1).water_ESM/analysis(1).initial_ESM;
analysis(1).labor_ESM =
SensitivityAnalysis_CropESM('potato','Infrastructure_Labor');
analysis(1).labor_change = analysis(1).labor_ESM/analysis(1).initial_ESM;

% For brown rice and tempeh, we must recalculate all crops under each case
% since changes to lower tier ingredients build on each other. Thus, we run
% a slightly altered version of Main.m to see what happens when we increase
% the cost of power by 20%:

% Read in crop and process lists.

[cropListNum,cropListTxt,cropListRaw] = xlsread('Crops/crop_list.xlsx');
[processListNum,processListTxt,processListRaw] =
xlsread('Processing/processing_data.xlsx');
```

```
% From processing data, find our maximum process tier.

maximum_tier = 0;

for j = 1:length(processListNum)
    if processListNum(j,3) > maximum_tier
        maximum_tier = processListNum(j,3);
    end
end

% Create a structure array to store processes by tier.
process_tier = struct();

% Create a counter variable for each tier (this will be used in the for
% loop that immediately follows).
for j = 1:maximum_tier
    process_tier(j).counter = 1;
end

% Populate the structure array with data from the list of processes.

for j = 1:length(processListNum)
    for k = 1:maximum_tier
        if processListNum(j,3) == k
            [process_tier(k).name(process_tier(k).counter)] =
processListRaw(j+1);
            process_tier(k).counter = process_tier(k).counter + 1;
        end
    end
end

% Create structure array to store data for crop and ingredient calculations
tieredESMlist = struct();

% Create structure array to store names of files with altered data.
new_infrastructure = struct();
new_infrastructure(1).file = 'Infrastructure_Power';
new_infrastructure(2).file = 'Infrastructure_Water';
new_infrastructure(3).file = 'Infrastructure_Labor';

% Now run our ESM loop function to populate the structure array with ESMs
% for all crops with new ESM cost of power, water and labor.
for k = 1:length(new_infrastructure)
    tieredESMlist(1).values =
SensitivityAnalysis_CropESMLoop(cropListTxt,new_infrastructure(k).file);

    % Run the ESM loop to populate the structure array with ESMs for all
    % process outputs with new ESM cost of power.
    for j = 1:maximum_tier
        tieredESMlist(j+1).values =
SensitivityAnalysis_ProcessedESMLoop(process_tier(j).name,tieredESMlist,new_i
nfrastructure(k).file);
    end
```

```
    % For readability, consolidate all the ESMs we have calculated
    % into one master list.
    ConsolidatedData_SensitivityAnalysis = ConsolidateESM(tieredESMlist);

    % Pull the data for brown rice and tempeh into our original structure
array:
    for j = 2:3
        for l = 1:length(ConsolidatedData_SensitivityAnalysis)
            if
strcmp(ConsolidatedData_SensitivityAnalysis(l).name,analysis(j).name)
                if strcmp(new_infrastructure(k).file,'Infrastructure_Power')
                    analysis(j).power_ESM =
ConsolidatedData_SensitivityAnalysis(l).ESM;
                    analysis(j).power_change =
analysis(j).power_ESM/analysis(j).initial_ESM;
                elseif
strcmp(new_infrastructure(k).file,'Infrastructure_Water')
                    analysis(j).water_ESM =
ConsolidatedData_SensitivityAnalysis(l).ESM;
                    analysis(j).water_change =
analysis(j).water_ESM/analysis(j).initial_ESM;
                elseif
strcmp(new_infrastructure(k).file,'Infrastructure_Labor')
                    analysis(j).labor_ESM =
ConsolidatedData_SensitivityAnalysis(l).ESM;
                    analysis(j).labor_change =
analysis(j).labor_ESM/analysis(j).initial_ESM;
                end
            end
        end
    end
end

% Results can be found in the structure array "analysis".

% Note that after you run this script, the values in tieredESMlist and
% consolidatedESMlist will no longer be valid - you must re-run Main.m to
% get the initial calculations.
```

G. CropGrowthArea.m

```
% This script takes a list of ingredients and desired quanitites of those
% ingredients, and calculates the total number of crop area in square
% meters required to grow the crops used to create those ingredients.

% First, read in our initial list of ingredients from an excel file.
[ingredientListNum,ingredientListTxt,ingredientListRaw] =
xlsread('sample_ingredient_list.xlsx');

% Create a structure array to store these ingredients and corresponding
% quantities, and populate it with data from the spreadsheet.
initial_ingredients = struct();

for j = 1:length(ingredientListNum)
    initial_ingredients(j).name = ingredientListRaw(j+1,1);
    initial_ingredients(j).quantity = ingredientListNum(j);
end

% We will need to iterate through the PreviousTier function until we have
% all integredients of Tier 0 (i.e. crops). So, create a structure array to
% store the data for each iteration.
iteration = struct();

% Create our "base case" by running the initial ingredient list through the
% PreviousTier function.
iteration.loop(1) = PreviousTier(initial_ingredients);

% Now use a while loop to keep iterating through PreviousTier until all our
% ingredients are tier 0.
k = 2;

while iteration.loop(k-1).highest_tier > 0
    iteration.loop(k) = PreviousTier(iteration.loop(k-1).inputs);
    k = k + 1;
end

% Now that we have all the crops necessary to create our ingredient, go
% back to the crop data and calculate the amount of square meters it will
% take to grow these crops. Store this data in a new structure array.

% Import our mission data from excel (we need this to calculate the total
% amount needed).
[missionNum,missionTxt,missionRaw] = xlsread('MissionData.xlsx');
crew_size = missionNum(1);

% Import our crop data from excel. Store the data we need in a new
% structure array called "crop_calculations" for clarity.

final_iteration = length(iteration.loop);
number_of_crops = length(iteration.loop(final_iteration).inputs);
```

```
for k = 1:number_of_crops
    crop = char(iteration.loop(final_iteration).inputs(k).name);
    fileExtension = '.xlsx';
    fileFolder = 'Crops/';
    fileName = strcat(crop,fileExtension);
    filePointer = strcat(fileFolder,fileName);

    [cropNum,cropTxt,cropRaw] = xlsread(filePointer);

    crop_calculations(k).name =
char(iteration.loop(final_iteration).inputs(k).name);
    crop_calculations(k).amount_needed =
iteration.loop(final_iteration).inputs(k).quantity;

    for l = 1:length(cropRaw)
        if strcmp(cropRaw(l,1),'plant_productivity')
            crop_calculations(k).plant_productivity = cropNum(l-2);
        end
    end
end

% Now using all of this data, calculate the areas needed in square meters
% to grow the desired quantities of each crop. Sum all of these up to get
% the total area in square meters required for the full ingredient list.

total_area = 0;

for l = 1:length(crop_calculations)
    crop_calculations(l).area = ((crew_size *
crop_calculations(l).amount_needed)/7) /
crop_calculations(l).plant_productivity;
    total_area = total_area + crop_calculations(l).area;
end

% Return the total area in square meters.
disp(total_area)
```

H. PreviousTier.m

```
function PreviousTier = PreviousTier(list)

% This function reads in a list of ingredients and desired quantities of
% each ingredient, and backtracks one tier to give the ingredients and
% quantities of ingredients required to make our initial list using at most
% one processing step per ingredient. (If an ingredient on our list is a
% crop, then the function will simply return the crop and its original
% quantity for that ingredient.)

% Read in processing data (will be used to calculate quantities of crops
% needed to create the ingredients):
[processingDataNum,processingDataTxt,processingDataRaw] =
xlsread('Processing/processing_data.xlsx');

% Read in list of crops (will be used to see whether ingredients are crops
% later in the program).
[cropNum,cropTxt,cropRaw] = xlsread('Crops/crop_list.xlsx');

% Create a structure array to store all the ingredients and the quantities
% we want of each one.
ingredients = list;

% Create structure array to store the inputs of our processing function
% (i.e. the inputs that we need to create the ingredient list we provided).
required = struct();

% Create a counter to count the number of required ingredients and crops.
m = 1;

% Keep track of highest process tier used to create our current outputs.
% (This will be use to judge whether our list has ultimately been reduced
% to a list of crops.)
required.highest_tier = 0;

% Now go through the whole list of ingredients and search through the
% processing data to find out which processes are used to create them and
% what inputs go into these processes.
for j = 1:length(ingredients)

    % First check to see if the ingredient is a crop. If it is, then our
    % calculation for this ingredient stops here and we do not need to go
    % through the processing search step.
    isCrop = 0;

    crops_list = LoadList(cropRaw(:,1));
    crops_list = RemoveNaN(crops_list);

    for l = 2:length(crops_list)
        if strcmp(char(crops_list(l)),ingredients(j).name)
            required.inputs(m).name = crops_list(l);
```

```
            required.inputs(m).quantity = ingredients(j).quantity;
            required.inputs(m).tier = 0;
            m = m + 1;
            isCrop = 1;
        end
    end

    % If our ingredient is not a crop, then we need to search for it within
    % our processes to see which process was used to create it, and what
    % the inputs for that process were.

    if isCrop == 0
    for k = 1:length(processingDataNum)

        % Read in data for all processes, and store it in structure arrays
        % so we can use it for our calculations.
        process_name = char(processingDataRaw(k+1));
        fileExtension = '.xlsx';
        fileFolder = 'Processing/';
        fileName = strcat(process_name,fileExtension);
        filePointer = strcat(fileFolder,fileName);

        [processDataNum,processDataTxt,processDataRaw] =
xlsread(filePointer);

        process_inputs = LoadList(processDataRaw(:,8));
        process_inputs = RemoveNaN(process_inputs);

        process_input_ratios = LoadList(processDataRaw(:,9));
        process_input_ratios = RemoveNaN(process_input_ratios);

        process_outputs = LoadList(processDataRaw(:,10));
        process_outputs = RemoveNaN(process_outputs);

        process_output_ratios = LoadList(processDataRaw(:,11));
        process_output_ratios = RemoveNaN(process_output_ratios);

        % Loop through the processes to see which process was used to
        % create our ingredient. Record the inputs for this process and
        % determine how much is required of each inputs using the input
        % ratio for the input, the yield for the process, and the output
        % ratio for our desired output. Record the required quantity for
        % each input.
        for l = 2:length(process_outputs)
           if strcmp(process_outputs(l),'') == false
               if strcmp(process_outputs(l),ingredients(j).name)
                   for n = 2:length(process_inputs)
                       if strcmp(process_inputs(n),'') == false
                           required.inputs(m).name = process_inputs(n);
                           required.inputs(m).quantity =
(cell2mat(process_input_ratios(n))/(cell2mat(process_output_ratios(l))*proces
singDataNum(k,2))) * ingredients(j).quantity;
                           required.inputs(m).tier = processingDataNum(k,3) -
1;
                           if required.inputs(m).tier > required.highest_tier
```

```
                                required.highest_tier =
required.inputs(m).tier;
                            end
                            m = m + 1;
                        end
                    end
                end
            end
        end
    end
    end
end

% Return a structure array containing all required inputs and the required
% quantities of each one.
PreviousTier = required;

end
```